But I Only Changed One Line of Code!

Theron R. Leishman
Software Technology Support Center/TRW

Dr. David A. Cook
Software Technology Support Center/Shim Enterprises, Inc.

One of the basic concepts widely accepted as a software best practice is software configuration management. Although generally accepted, basic configuration management activities are often ignored, resulting in serious negative impact on software development and acquisition projects. This article is an introduction to basic, software configuration management concepts. It introduces these basic concepts and provides rationale for their implementation.

We are sitting in the Chicago O'Hare airport hours after the departure time of a flight home. After sitting on the plane for more than an hour, the pilot indicated that he was unable to get one of the engines to start. The pilot returned to the gate, and we were allowed to deplane. Eventually, it was determined that a hose in the engine was malfunctioning. This single hose was essential to the proper functioning of the engine. Fortunately, the engine mechanic was able to determine the hose that was causing the problem.

The mechanic understood the configuration of the plane engine and was able to refer to a defined list of all the engine parts, their relative arrangement to each other, and the methods to be used to assemble these parts into a jet engine. This is what is referred to as a configuration. The ability to properly distinguish the appropriate parts that must be used to build each type of engine is critical to assure consistent, safe production and use of aircraft engines.

As software developers, is it any less critical for us to be able to manage the assets that we create? The software we create is often maintained in various forms, with different tools, during its various stages of development, usage, maintenance, and operation. It is maintained at diverse locations, in diverse formats, and by diverse organizations. Software components are composed of parts, which are themselves software and are created using tools that are also software. Software systems are made of parts that cannot be touched, picked up, physically put in place, or manipulated. If you lose track of one of the software pieces, you have to re-create it. However, if you lose track of one of the tools, you *hope* that you can procure a new one. If the tool vendor is out of business – or no longer sells or supports the tool you need – you have a problem.

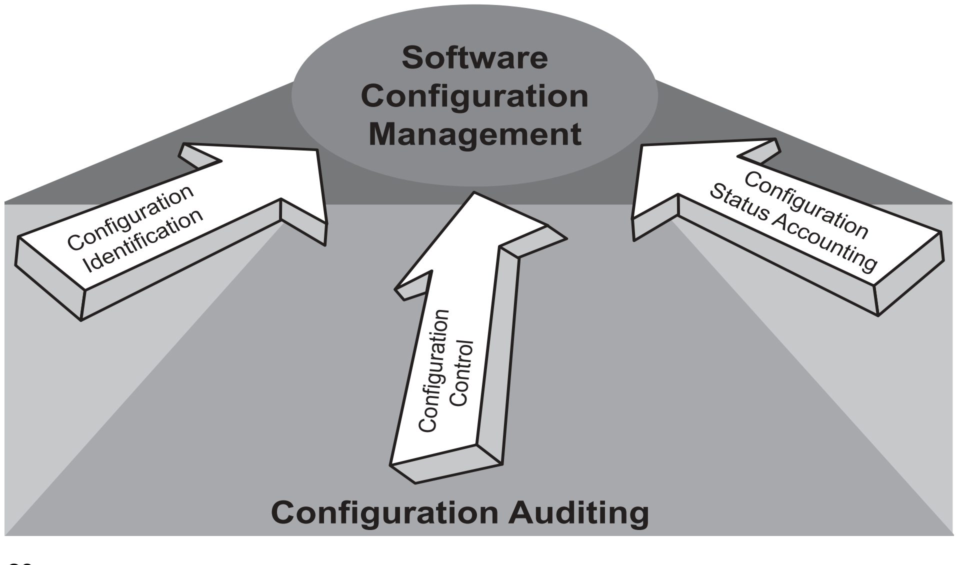
Software Configuration Management

Software Configuration Management (SCM) has been defined as the art of identifying, organizing, and controlling modifications to software [1]. SCM is a process that should be performed across the entire software development and maintenance life cycle. The configuration of software systems is by their nature complex. SCM may be described as a well-defined arrangement of software parts and the exact procedures and tools to be used for constructing the product or system from these parts. This must also include procedures for reconstructing various versions and releases [2]. The concepts of SCM are not new; numerous texts have been written on the topic of SCM. Typically SCM is defined by, and broken up into, the following four functional areas.

- Configuration Identification.
- Configuration Control.
- Configuration Status Accounting.
- Configuration Auditing.

SCM involves identifying what software assets are important to the organization. It includes controlling changes to these assets to ensure their integrity. Accounting for the status of these assets is important to the ongoing success of the organization. To assure that software assets are being properly accounted for, periodic configuration audits should be performed. Figure 1 depicts SCM's four basic functions. An introduction to each of these functions follows in this article.

Figure 1: *The Basic Functions of Software Configuration Management*


Software Configuration Management
Configuration Identification
Configuration Control
Configuration Status Accounting
Configuration Auditing

Configuration Identification

To properly manage all the elements of an aircraft engine, each of the critical parts must be properly identified. This is a key element in allowing proper management of the various engine configurations.

To provide the proper level of configuration management to software parts, we must also identify all the items that must be managed. These items, which we need to identify, are referred to as configuration items. Software configuration items (SCIs) are the items that are determined to be essential to manage the software product of concern. SCIs are the objects required to design, develop, build, maintain, test, and field a software product. SCIs are not things that can necessarily be shared between organizations, or even from one project to another. What an organization can, and should do, is develop documented criteria that can be consistently applied in determining which software-related items should be placed under configuration control.

Identification is the most important function of SCM. This is because items that are not identified cannot be managed. If you need *anything* to create your software product, then that item must be identified and controlled. This includes compilers, operating systems, libraries, development tools, or environments. It might even include manuals and other documentation. It could also include such items as current spreadsheet programs, database programs, and word processing programs. In short, if you need it to accomplish your development task, then you probably should assign an identification number to it.

Configuration Control

As an inexperienced programmer, one of the authors of this article was approached by the primary stakeholder of an application for which he held maintenance responsibility. The stakeholder, a very influential individual in the company, wanted to have a small change made to a section of the application. Being pushed by the stakeholder to make the change, and being anxious to make brownie points in the company, the author made the simple, one line code change.

It was not until 2:00 a.m. the next morning, during the middle of a production run, that it was determined that the one line change had brought the entire application down. Not only was this application down, but also a critical corporate system that required input from the application was unable to complete essential processing while waiting for input.

Ouch! The desire to make a good impression in the organization backfired! The problem was that neither the organization nor any developers were controlling software assets. Anyone was allowed to make changes to the software at will. They were then able to have the changes migrated into a production environment with little or no unit testing, to say nothing of integration testing.

Software configuration control is the process of controlling and limiting changes made to software assets. According to the Institute of Electrical and Electronics Engineers, configuration control is an element of configuration management consisting of the evaluation, coordination, approval or disapproval, and implementation of changes to configuration items after formal establishment of their configuration identification [3]. In our words, configuration control is making sure that changes to software assets are only allowed to occur after they have been analyzed, evaluated, reviewed, and approved by a group authorized to control changes to software assets.

This group of people act as gatekeepers to control the flow of changes made to a SCI. They have authority to approve or veto proposed changes. This group is known as a change control board or configuration control board (CCB). The primary role of the CCB is to ensure that every requested change to an SCI is properly considered and coordinated prior to it being incorporated. The CCB should include representation from program management, systems engineering, software engineering, software quality assurance, software configuration management, independent test, and a customer representative.

The CCB is responsible for seeing that all change requests are adequately reviewed, understood, and analyzed for impact prior to their decision to accept, defer to the next release, or decline the change request. Before any change to a SCI is made, the CCB ensures that someone did the research necessary to verify that the change will not unexpectedly impact other SCIs. The CCB, to make its decision, evaluates the impacts of each proposed change, and decides if the change is worth making.

Figure 2 presents a sample software change control process. A software design review board prior to the development of an engineering change proposal (ECP) reviews requested changes, enhancements, or problem reports. Once developed, the software CCB then evaluates ECP's. This board controls which ECP's are approved for incorporation into the software. Following approval and prioritization the change/enhancement will be made to the software.

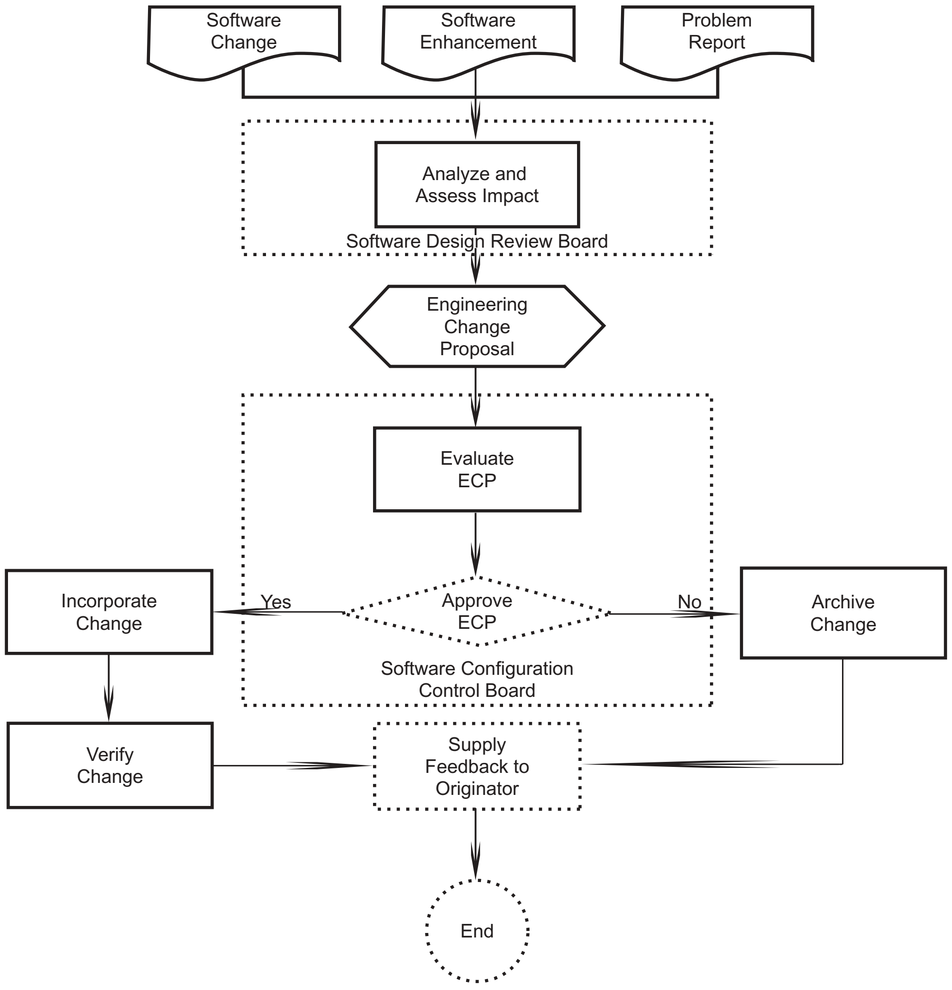

Software Change
Software Enhancement
Problem Report
Analyze and Assess Impact
Software Design Review Board
Engineering Change Proposal
Evaluate ECP
Approve ECP
Yes
No
Incorporate Change
Archive Change
Software Configuration Control Board
Verify Change
Supply Feedback to Originator
End

Figure 2: *The Software Change Process*

Configuration Status Accounting

While attending college, one of the authors of this article was required to endure a course on cost accounting. In this course, he learned that cost accounting is that part of accounting that identifies, defines, measures, reports, and analyzes the various elements of direct and indirect costs associated with manufacturing and providing goods or services. Companies tend to care about things (assets) for which they have a financial investment.

Cost accountants are expected to track these corporate assets. They must know what the assets are, what costs were incurred in the development and ongoing maintenance of the asset, how the asset works with or interfaces with other assets, and be able to perform analysis on and report the status of these assets. To properly account for an organization's assets, accountants must track the cost of each asset. In a manufacturing environment this may include assets that roll into other assets. The work, or product breakdown must be documented and all items tracked.

SCI's are the assets of a software product and must likewise be accounted for. Status accounting is the SCM activity responsible for tracking and maintaining data relative to each of these SCI's. It includes the tracking of changes to SCI's, and provides the ability to determine the status of each item at any phase of the software development or maintenance process. Status accounting involves gathering information to answer the following questions:

- What changes have been requested?
- What changes have been made?
- When did each change occur?
- What was the reason for each change?
- Who authorized each change?
- Who performed each change?
- What SCIs were affected by each change?

A repository of key software assets must be maintained to ensure that these key assets are properly accounted for. Changes to the status of these assets must be documented and tracked to allow the organization to know the status of any software asset at any point in time. Reports indicating the configuration of these software products must be defined and generated as a critical part of the accounting process. Following is a list of sample reports that the status accounting function should provide.

- Transaction log.
- Change log.
- SCI delta report.
- Resource usage report.
- SCI status report.
- Changes in progress report.
- Change request status report.
- Change completion report by developer, application, etc.

Configuration Auditing

There has been much interest lately in the integrity of certain large accounting firms who perform *independent* audits of corporate books. These audits are intended to attest to the soundness of the financial information reported by corporations.

In a previous lifetime, one of the authors of this article had the *opportunity* to work as an internal auditor. In this role he performed compliance audits, operational audits, proposal audits, and other types of audits in support of the mission of the organization. He was labeled by one acquaintance as Mr. Sneak & Peak. Auditors are not widely embraced as *good guys*. They are more likely considered those who go in after the war is over and stab the wounded.

By auditing adherence to policies, processes, and procedures, this corporation was able to demonstrate to their government customer that sound practices were consistently being followed in the delivery of the products and services for which the government was paying. In addition, people are more likely to follow the approved practices if they know someone is checking to see that they do. If developers gather metrics that nobody ever evaluates, the developers quickly learn that the metrics can safely be ignored. By the same token, if developers are never held accountable, then they quickly learn that policies, processes and procedures can be ignored. There must be accountability.

Software configuration audits are important for the same reasons. It is generally accepted that following good software development practices will contribute to consistent delivery of quality software products (assets). Audits should be conducted to help assure that software development policies, processes, and procedures are being consistently followed and adhered to.

In "Software Configuration Management for Project Leaders" presented at the 1997 Software Technology Conference in Salt Lake City, Tim Kasse explained that configuration audits verify that the software product is built according to the requirements, standards, or contractual agreement. Auditing also verifies that all software products have been produced, correctly identified, described, and that all requested changes are resolved.

A software configuration audit should be periodically performed to ensure that the SCM practices and procedures are rigorously followed. The integrity of the software baseline must be assessed. The completeness and correctness of the software baseline library contents must be verified. The accuracy of changes to the baselines must be verified to ensure that the changes were implemented as intended. It is recommended that a software configuration audit be performed before every major baseline change. Software configuration auditing should be continuous, with increased frequency and depth throughout the life cycle [4].

With regard to SCM, the terms functional configuration audit and physical configuration audit are often used. A functional configuration audit is intended to verify that the software functions as defined by the software requirements documentation. A physical configuration audit is intended to verify that all the items identified to be included in software release are actually included and that no additional items are included.

Who Cares?

OK, we have said lots of nice words about SCM, but why do you care about all this? Findings from various software assessments indicate that the lack of adherence to basic software development and acquisition sound practices continues to have a negative cost and schedule impact of software development, maintenance, and acquisition projects. SCM is one of these sound practices. Watts Humphrey said:

> The most frustrating software problems are often caused by poor configuration management. The problems are frustrating because they take time to fix, they often happen at the worst time, and they are totally unnecessary. [5]

If any of the following situations sound familiar to you, then you care!

- A software bug that was fixed six months ago has suddenly reappeared.
- A programmer just spent 12 hours making a change to the wrong version of the software.
- There is no way to trace requirements from the requirements document to the user documentation and source code.
- Two programmers working on a project have overwritten each other's code, rendering the last 40 hours of each of their work useless.
- No one can find the latest version of the source code.
- Fielded software that was working fine yesterday mysteriously will not work today.
- An application installed at various locations is running fine at some locations, but not at other locations.
- There is no way to evaluate impact on requirements from proposed changes.

Conclusion

In short, there is no such thing as a one-line change! Any software change is based on some type of requirement change. Proper identification of all the items critical to the development of software to satisfy requirements is essential to meeting those changing requirements. A supposed one-line change requires proper identification of what actually needs to change. It requires analysis of how the piece of code affected interfaces with other sections of the code. It requires analysis of what impact a change to the code will have on other sections of the code. It requires review and approval by individuals who know and understand the application and can determine the extended impact of the requested change. It also requires documentation and traceability into what change is really required, who approved the change, when the change was made, and why it was made. It also assumes that someone is watching the process, that audits are being conducted assuring that the approved process for making changes is being followed, and that the software product matches the documentation.

Applying the concepts of SCM will improve the organization's ability to control software assets. Items and components essential to the development and maintenance of the software will be identified, managed, and controlled. Changes will not be made without proper analysis and approval. The status of software assets will be known and traceable at all times, and periodic audits will assure that the process is being followed. The probability of simple software changes hugely impacting projects will be greatly reduced.◆

References

1. Babich, Wayne A. Software Configuration Management. Mass.: Addison-Wesley, 1986.
2. Ben-Menachem, Mordechai. Software Configuration Management Guidebook. London: McGraw-Hill, 1994.
3. Institute of Electrical and Electronics Engineers. "Glossary of Software Engineering Terminology." IEEE Std-610.12-1990. New York: IEEE, 1993.
4. Kasse, Tim. Proceedings of The 9th Annual Software Technology Conference. Salt Lake City, UT. Software Technology Conference, 1997.
5. Humphrey, Watts. Managing the Software Process. Addison-Wesley, 1989.

About the Authors

Theron R. Leishman is a consultant currently on contract with the Software Technology Support Center at Hill Air Force Base, Utah. Leishman has 18 years experience in various aspects of software development. He has successfully managed software projects and performed consulting services for the Department of Defense, aerospace, manufacturing, health care, higher education, and other industries. This experience has provided a strong background in systems analysis, design, development, project management, and software process improvement. Leishman has a master's degree in business administration from the University of Phoenix. He is a Level 2 Certified International Configuration Manager by the International Society of Configuration Management, and is employed by TRW.

Software Technology Support Center
7278 4th Street
Bldg. 100
Hill AFB, UT 84056
Phone: (801) 775-5738
Fax: (801) 777-8069
E-mail: theron.leishman@hill.af.mil

David A. Cook, Ph.D., is the principal engineering consultant, Shim Enterprises, Inc. He is currently assigned as a software-engineering consultant to the Software Technology Support Center at Hill Air Force Base, Utah. Dr. Cook has more than 27 years of experience in software development and software management. He was formerly an associate professor of computer science at the U. S. Air Force Academy and also the deputy department head of the Software Professional Development Program at the Air Force Institute of Technology. Dr. Cook has published numerous articles on software process improvement, software engineering, object-oriented software development, programming languages, and requirements engineering. He has a doctorate degree in computer science from Texas A&M University, and he is an authorized Personal Software Process instructor.

Software Technology Support Center
7278 4th Street
Bldg. 100
Hill AFB, UT 84056
Phone: (801) 775-3055
Fax: (801) 777-8069
E-mail: david.cook@hill.af.mil

CrossTalk
The Journal of Defense Software Engineering

Get Your Free Subscription

Fill out and send us this form.

OO-ALC/MASE
7278 Fourth Street
Hill AFB, UT 84056-5205
Fax: (801) 777-8069 DSN: 777-8069
Phone: (801) 775-5555 DSN: 775-5555

Or request online at www.stsc.hill.af.mil

NAME:____________________

RANK / GRADE:____________________

POSITION / TITLE:____________________

ORGANIZATION:____________________

ADDRESS:____________________

____

BASE / CITY:____________________

STATE:__________ZIP:__________

PHONE:(____)____________________

FAX:(____)____________________

E-MAIL:____________________

CHECK BOX(ES) TO REQUEST BACK ISSUES:

JUL2001	☐	TESTING & CM
AUG2001	☐	SW AROUND THE WORLD
SEP2001	☐	AVIONICS MODERNIZATION
JAN2002	☐	TOP 5 PROJECTS
MAR2002	☐	SOFTWARE BY NUMBERS
MAY2002	☐	FORGING THE FUTURE OF DEF.
JUN2002	☐	SOFTWARE ESTIMATION
AUG2002	☐	SOFTWARE ACQUISITION
SEP2002	☐	TEAM SOFTWARE PROCESS
OCT2002	☐	AGILE SOFTWARE DEV.
NOV2002	☐	PUBLISHER'S CHOICE
DEC 2002	☐	YEAR OF ENG. AND SCI.

To Request Back Issues on Topics Not Listed Above, Please Contact Karen Rasmussen at <karen.rasmussen@hill.af.mil>.